```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, Model
from tensorflow.keras.datasets import mnist, imdb
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# Set random seed for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# --- Q1: Basic Autoencoder ---
print("Q1: Implementing a Basic Autoencoder")

# Load and preprocess MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = x_train.reshape(-1, 784)  # Flatten to (28*28 = 784)
x_test = x_test.reshape(-1, 784)

# Function to build autoencoder with variable latent dimension
def build_autoencoder(latent_dim):
    # Encoder
    inputs = layers.Input(shape=(784,))
    encoded = layers.Dense(latent_dim, activation='relu')(inputs)
    # Decoder
    decoded = layers.Dense(784, activation='sigmoid')(encoded)
    autoencoder = Model(inputs, decoded)
    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
    return autoencoder

# Train autoencoders with different latent dimensions
latent_dims = [16, 32, 64]
reconstructions = {}
for dim in latent_dims:
    print(f"Training autoencoder with latent dim: {dim}")
    autoencoder = build_autoencoder(dim)
    autoencoder.fit(x_train, x_train, epochs=10, batch_size=128, validation_data=(x
    reconstructions[dim] = autoencoder.predict(x_test)

# Plot original vs reconstructed images
plt.figure(figsize=(15, 5))
```

```
plt.figure(figsize=(15, 5))
for i, dim in enumerate(latent_dims):
    plt.subplot(3, 3, i*3 + 1)
    plt.imshow(x_test[0].reshape(28, 28), cmap='gray')
    plt.title("Original")
    plt.axis('off')
    plt.subplot(3, 3, i*3 + 2)
    plt.imshow(reconstructions[dim][0].reshape(28, 28), cmap='gray')
    plt.title(f"Latent Dim: {dim}")
    plt.axis('off')
plt.tight_layout()
plt.savefig('q1_reconstructions.png')
plt.close()

# --- Q2: Denoising Autoencoder ---
print("Q2: Implementing a Denoising Autoencoder")

# Add Gaussian noise to training and test data
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

# Build and train denoising autoencoder (latent dim = 32)
denoising_autoencoder = build_autoencoder(32)
denoising_autoencoder.fit(x_train_noisy, x_train, epochs=10, batch_size=128, valida
reconstructed_noisy = denoising_autoencoder.predict(x_test_noisy)

# Plot noisy vs reconstructed images
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.imshow(x_test[0].reshape(28, 28), cmap='gray')
plt.title("Original")
plt.axis('off')
plt.subplot(1, 3, 2)
plt.imshow(x_test_noisy[0].reshape(28, 28), cmap='gray')
plt.title("Noisy")
plt.axis('off')
plt.subplot(1, 3, 3)
plt.imshow(reconstructed_noisy[0].reshape(28, 28), cmap='gray')
plt.title("Reconstructed")
plt.axis('off')
plt.tight_layout()
plt.savefig('q2_denoising.png')
plt.close()
```

```
# --- Q3: RNN for Text Generation with The Little Prince ---
print("Q3: Implementing an RNN for Text Generation (The Little Prince)")

# Sample text from "The Little Prince" by Antoine de Saint-Exupéry
text = """
it is only with the heart that one can see rightly;
what is essential is invisible to the eye.
""".lower().strip()
chars = sorted(list(set(text)))
char_to_idx = {c: i for i, c in enumerate(chars)}
idx_to_char = {i: c for i, c in enumerate(chars)}
vocab_size = len(chars)

# Prepare sequences
max_len = 10
step = 1
sequences = []
next_chars = []
for i in range(0, len(text) - max_len, step):
    sequences.append(text[i:i + max_len])
    next_chars.append(text[i + max_len])
X = np.zeros((len(sequences), max_len, vocab_size), dtype=np.bool_)
y = np.zeros((len(sequences), vocab_size), dtype=np.bool_)
for i, seq in enumerate(sequences):
    for t, char in enumerate(seq):
        X[i, t, char_to_idx[char]] = 1
    y[i, char_to_idx[next_chars[i]]] = 1

# Build RNN model
model_rnn = tf.keras.Sequential([
    layers.LSTM(128, input_shape=(max_len, vocab_size)),
    layers.Dense(vocab_size, activation='softmax')
])
model_rnn.compile(optimizer='adam', loss='categorical_crossentropy')
model_rnn.fit(X, y, epochs=50, batch_size=32, verbose=0)

# Text generation function with temperature
def generate_text(model, seed, length=50, temperature=1.0):
    generated = seed
    for _ in range(length):
        x_pred = np.zeros((1, max_len, vocab_size))
        for t, char in enumerate(seed[-max_len:]):
            x_pred[0, t, char_to_idx[char]] = 1
        preds = model.predict(x_pred, verbose=0)[0]
```

```
        preds = np.log(preds + 1e-10) / temperature  # Apply temperature
        next_idx = np.random.choice(range(vocab_size), p=np.exp(preds) / np.sum(np.
        next_char = idx_to_char[next_idx]
        generated += next_char
        seed = generated[-max_len:]
    return generated

# Generate text with different temperatures
seed = "it is only"
print("Generated text (temp=0.5):", generate_text(model_rnn, seed, temperature=0.5)
print("Generated text (temp=1.0):", generate_text(model_rnn, seed, temperature=1.0)

# --- Q4: Sentiment Classification Using RNN ---
print("Q4: Sentiment Classification Using RNN")

# Load IMDB dataset
max_features = 5000
maxlen = 100
(x_train_imdb, y_train_imdb), (x_test_imdb, y_test_imdb) = imdb.load_data(num_words
x_train_imdb = tf.keras.preprocessing.sequence.pad_sequences(x_train_imdb, maxlen=m
x_test_imdb = tf.keras.preprocessing.sequence.pad_sequences(x_test_imdb, maxlen=max

# Build sentiment classification model
model_sentiment = tf.keras.Sequential([
    layers.Embedding(max_features, 128, input_length=maxlen),
    layers.LSTM(64),
    layers.Dense(1, activation='sigmoid')
])
model_sentiment.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc
model_sentiment.fit(x_train_imdb, y_train_imdb, epochs=3, batch_size=64, validation

# Evaluate model
y_pred = (model_sentiment.predict(x_test_imdb) > 0.5).astype(int)
cm = confusion_matrix(y_test_imdb, y_pred)
print("Confusion Matrix:\n", cm)
print("Classification Report:\n", classification_report(y_test_imdb, y_pred))

# Plot confusion matrix
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.savefig('q4_confusion_matrix.png')
plt.close()

print("Assignment completed! Check saved plots and outputs.")
```

```
Q1: Implementing a Basic Autoencoder
Training autoencoder with latent dim: 16
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Training autoencoder with latent dim: 32
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Training autoencoder with latent dim: 64
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Q2: Implementing a Denoising Autoencoder
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Q3: Implementing an RNN for Text Generation (The Little Prince)
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserW
  super().__init__(**kwargs)
Generated text (temp=0.5): it is only wihh shhthttttttt  . ..y..ewwhhttthh  tar
Generated text (temp=1.0): it is only witht e e e..hyyryawtht  t neeree hyt
yataa  o se
Q4: Sentiment Classification Using RNN
Epoch 1/3
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90:
  warnings.warn(
391/391 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - accuracy: 0.7172 - loss: 0.5310 -
Epoch 2/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 6s 14ms/step - accuracy: 0.8719 - loss: 0.3063 -
Epoch 3/3
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 14ms/step - accuracy: 0.8922 - loss: 0.2615 -
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step
Confusion Matrix:
 [[ 9753  2747]
 [ 1221 11279]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.78      0.83     12500
           1       0.80      0.90      0.85     12500

    accuracy                           0.84     25000
   macro avg       0.85      0.84      0.84     25000
weighted avg       0.85      0.84      0.84     25000

Assignment completed! Check saved plots and outputs.
```